Trabajo Práctico 2: Programación Lógica

Paradigmas de Lenguajes de Programación — 2do cuat. 2017

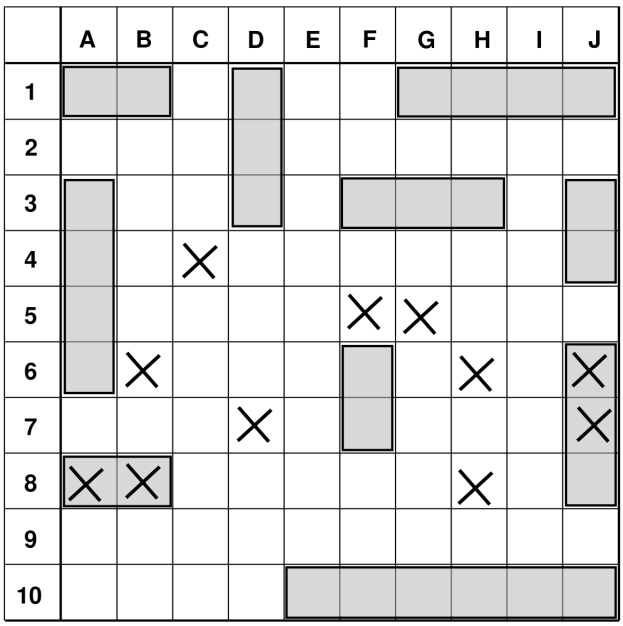
Fecha de entrega: 17 de octubre de 2017

1. Introducción

Deseamos implementar en Prolog una batalla naval. La misma consiste en un juego para 2 jugadores, con un tablero de N × M por cada uno de ellos y una serie de barcos que ubicará cada jugador en su tablero.
Los barcos deben colocarse en dirección horizontal o vertical, y deben ubicarse de manera tal que no se toquen entre sí. Es decir, no pueden compartir celdas, ni ubicarse en celdas adyacentes (ya sea en dirección horizontal, vertical o diagonal).
El objetivo del juego es hundir la flota del contrincante. Para ello, debe colocar su propia flota de forma estratégica y encontrar y hundir con los disparos la flota contraria.
Cada jugador dispone de un turno de disparo que se irá alternando. Para hacerlo dirá las coordenadas en las que quiere atacar, como por ejemplo fila 1, columna 3.
Al disparar, el otro jugador comprobará el resultado en su tablero:

- Si la casilla está en blanco, responderá "agua".
- Si en la casilla se encuentra parte de un barco responderá "tocado". En ese caso el jugador tiene derecho a un nuevo disparo en el mismo turno.
- Si en la casilla se encuentra un barco de un cuadro o la última parte de un barco ya tocado, responderá "hundido" y también tiene derecho a un nuevo disparo.

Durante el transcurso del juego, cada jugador anota en un cuadro los resultados de sus ataques a la flota enemiga. Finalmente, gana el jugador que antes consigue hundir la flota del otro.
En este TP modelaremos todo lo necesario para iniciar el juego y realizar ataques.



Fase de preparación

Los siguientes predicados permitirán preparar el juego colocando los barcos en el tablero, el cual se representará como una matriz (lista de filas) cuyas posiciones empiezan en 1. Inicialmente, la matriz contendrá variables libres en todas sus posiciones, que luego se irán llenando con barcos.
Utilizaremos el átomo **o** para representar las posiciones ocupadas por un barco.
Ya está definido el predicado `matriz(?Matriz, ?Filas, ?Columnas)` que, dada una matriz, instancia en `Filas` y `Columnas` su dimensión o, dada una dimensión, genera una matriz de dicha dimensión con variables libres en todas sus posiciones. Esta matriz será la base del tablero de juego.

Nota sobre instanciación: además de las anotaciones `+`, `-` y `?` ya conocidas, en este TP se introduce la anotación `+?`. Esta indica que el argumento puede estar total o parcialmente instanciado y que, en caso de contener variables, una o más de ellas podrían instanciarse producto del predicado.

Ejercicio 1

Implementar el predicado `contenido(+?Tablero,?Fila,?Columna,?Contenido)`, que tiene éxito si `Contenido` es el contenido de la celda en la fila `Fila` y columna `Columna` en el tablero `Tablero`.

Ejercicio 2

Implementar el predicado `disponible(+Tablero,?Fila,?Columna)`, que indica si una celda está disponible para colocar un barco (o parte de un barco). Una celda está disponible si tanto ella como todas las que la rodean contienen variables no instanciadas.
Sugerencia: usar el predicado `adyacenteEnRango(+Tablero,+F1,+C1,?F2,?C2)`, que es verdadero cuando la posición `(F2,C2)` es adyacente a `(F1,C1)` y está en rango en el tablero.

Ejercicio 3

Definir el predicado `puedoColocar(+CantPiezas,?Direccion,+Tablero,?Fila,?Columna)`, que es verdadero cuando `Direccion` es el átomo `vertical` u `horizontal`, y `Fila` y `Columna` representan una posición del tablero a partir de la cual se puede colocar un barco de `CantPiezas` piezas en la dirección `Direccion`.
Por ejemplo[1]:

```
?- matriz(M,2,4), puedoColocar(3,Dir,M,F,C).
M = [[_, _, _, _], [_, _, _, _]],
Dir = horizontal,
F = C, C = 1 ;
M = [[_, _, _, _], [_, _, _, _]],
Dir = horizontal,
F = 1,
C = 2 ;
M = [[_, _, _, _], [_, _, _, _]],
Dir = horizontal,
F = 2,
C = 1 ;
M = [[_, _, _, _], [_, _, _, _]],
Dir = horizontal,
F = C, C = 2 ;
false.
```

```
?- matriz(M,2,3), contenido(M,2,1,o), puedoColocar(2,Dir,M,F,C).
M = [[_, _, _], [o, _, _]],
Dir = vertical,
F = 1,
C = 3 ;
false.
```

Sugerencia: escribir un predicado que dadas una dirección y la fila y columna de una celda, instancie la fila y columna de la siguiente celda en esa dirección.

Ejercicio 4

Definir el predicado `ubicarBarcos(+Barcos,+?Tablero)` que, siendo `Tablero` un tablero parcialmente instanciado y `Barcos` una lista con las longitudes (cantidad de piezas) de los barcos a ubicar,

[1] Las variables no instanciadas se muestran como _ en los ejemplos por claridad.

instancia las posiciones libres que correspondan en el tablero, colocando todos los barcos del tablero en ubicaciones válidas.
Recordar que las piezas de los barcos se representan con el átomo **o**, y que los barcos no deben tocarse entre sí.
Por ejemplo:

```
?- matriz(M,3,2), ubicarBarcos([2,1],M).
M = [[o, o], [_, _], [o, _]] ;
M = [[o, o], [_, _], [_, o]] ;
M = [[o, _], [_, _], [o, o]] ;
M = [[_, o], [_, _], [o, o]] ;
false.
```

Fase de ataque

Una vez ubicados los barcos en el tablero, se deberá completar el tablero rellenando las celdas que quedaron libres con el átomo ∼, que representa el agua. Una vez completo el tablero, se podrá empezar a atacar la flota.

Ejercicio 5

Implementar el predicado `completarConAgua(+?Tablero)`, que dado un tablero parcialmente instanciado, instancia todas sus celdas libres con el átomo ∼.
Por ejemplo:

```
?- Tablero = [[o, o], [_, _], [_, o]], completarConAgua(Tablero).
Tablero = [[o, o], [∼, ∼], [∼, o]] ;
false.
```

Sugerencia: usar `maplist/2` para aplicar un predicado a todas las filas del tablero.

Ejercicio 6

Definir el predicado `golpear(+Tablero,+NumFila,+NumColumna,-NuevoTab)`, que es verdadero cuando `numFila` y `numColumna` son los índices de una fila y columna del tablero `Tablero`, y `NuevoTab` es igual a `Tablero` excepto porque en la posición (`numFila`, `numColumna`) tiene agua. Si `Tablero` ya tenía agua en esa posición, entonces ambos tableros son iguales.

Ejercicio 7

Definir el predicado `atacar(+Tablero,+NumFila,+NumColumna,-Resultado,-NuevoTab)`,
que es verdadero cuando `numFila` y `numColumna` son los índices de una fila y columna del tablero `Tablero`, `Resultado` es el átomo `agua`, `tocado` o `hundido` según hubiera agua, parte de un barco no atacada o un barco de una sola pieza - o la última pieza que queda de un barco - en la posición (`numFila`, `numColumna`) de `Tablero`, y `NuevoTab` es el tablero que queda después de haber atacado la posición correspondiente. Por ejemplo:

```
?- atacar([[o, o], [∼, ∼], [∼, o]],1,1,Res,T).
Res = tocado,
T = [[∼, o], [∼, ∼], [∼, o]] ;
false.

?- atacar([[o, o], [∼, ∼], [∼, o]],3,1,Res,T).
Res = agua,
T = [[o, o], [∼, ∼], [∼, o]] ;
false.

?- atacar([[o, o], [∼, ∼], [∼, o]],3,2,Res,T).
Res = hundido,
T = [[o, o], [∼, ∼], [∼, ∼]] ;
false.
```

Ejercicio 8

El predicado del ejercicio anterior, ¿es reversible en alguno de sus parámetros? Escribir en los comentarios qué parámetros son reversibles, cuáles no y por qué.

2. Pautas de Entrega

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado.
Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp2.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- utilización de unificación, backtracking, generate and test y reversibilidad de los predicados.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

⚠ Se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

3. Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía → Programación Lógica*).
Se recomienda que utilicen los predicados ISO y los de SWI-Prolog ya disponibles, siempre que sea posible. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Cosas útiles* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.